

ECMWF's IFS
on an IBM p690 system with
960 Power4 processors

Deborah Salmond

Agenda

➤ ECMWF

- who we are and what we do

➤ Supercomputers at ECMWF

- CRAY to Fujitsu to IBM

- Procurement in 2001

➤ IFS (Integrated Forecast System)

- Optimisation of algorithm

- Design for vector, scalar, MPP and SMP

- MPI and OpenMP

➤ Migration from Fujitsu VPP5000 to IBM p690

- Performance comparisons

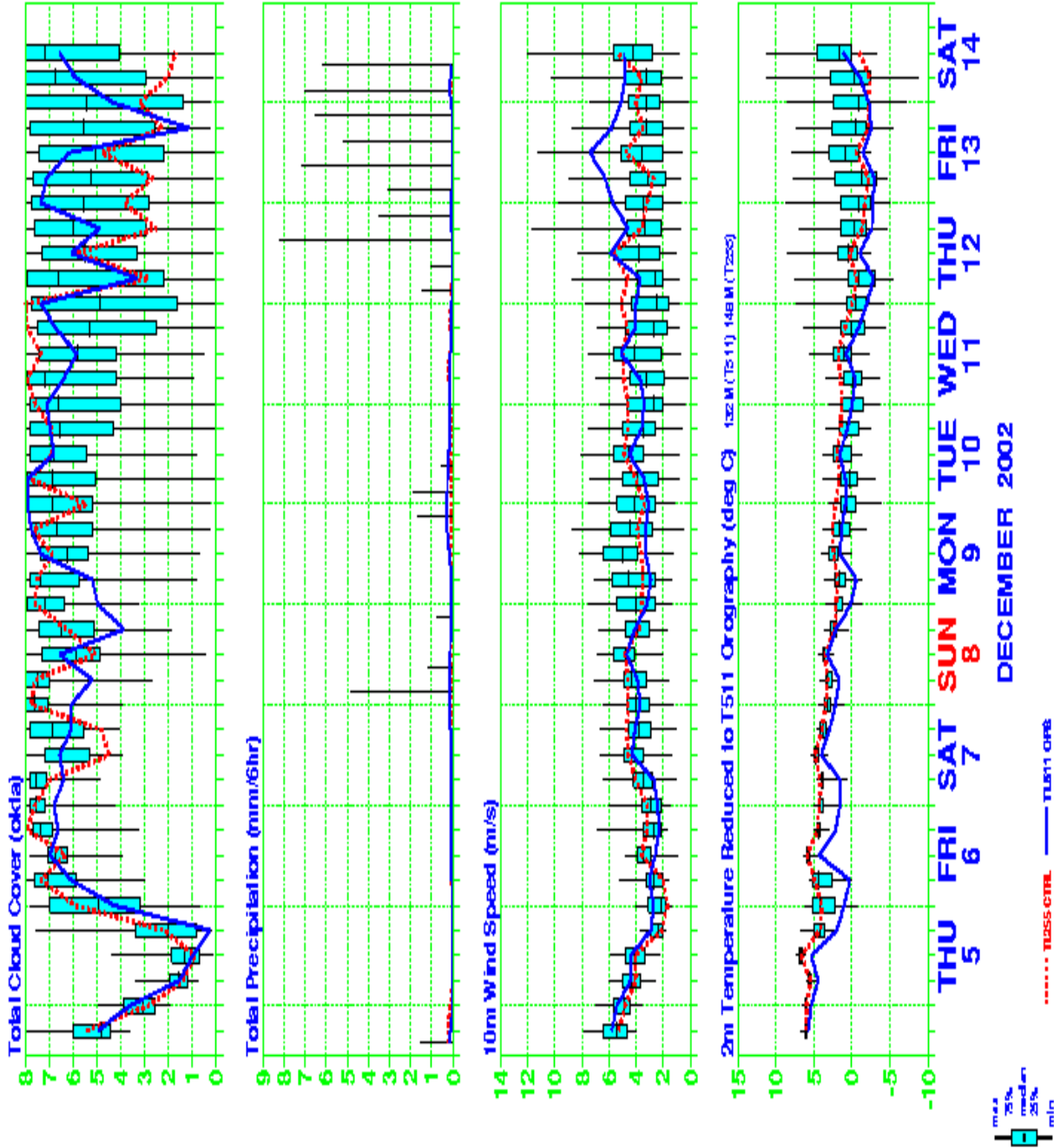
➤ Optimisation for IBM p690

ECMWF

European Centre for Medium-range Weather Forecasts

- Supported by 23 European states
- Founded in 1975
- 10 day Global Weather Forecast
 - High resolution atmospheric model and data assimilation
- EPS (Ensemble Prediction System)
 - Probability that we got it right

EPS Meteorogram
 Daresbury 53.0° N 3.0° W
 Deterministic Forecasts and EPS Distribution 4 December 2002 12 UTC



ECMWF Operational forecasting system

- Data assimilation
 - Four-dimensional variational system (4D-Var)
- Global atmospheric forecasts
 - 10 day Forecasts at 40 km resolution (T511)
 - Ensemble of 51 forecasts at 80 km resolution (T255)
- Ocean wave forecasts
 - 10 day Global forecasts at 40 km resolution
 - 5 day European forecasts at 15 km resolution
- Global Seasonal forecasts
 - 6 month forecasts (ensemble with 30 members) coupled atmosphere & ocean

Planned improvements to forecasting system

- More satellite observation data
 - More vertical levels
 - Severe Weather forecasting system
- More computing power required

ECMWF supercomputers

1977	CRAY1	Vector
	CRAY XMP-2	} Vector + Shared Memory Parallel
	CRAY XMP-4	
	CRAY YMP-8	
	CRAY C90-16	
	Fujitsu VPP700	} Vector + MPI Parallel
	Fujitsu VPP5000	
2002	IBM p690	Scalar + MPI + Shared Memory Parallel

UKMO supercomputers

1970	IBM 360/195	Scalar
	CYBER 205	Vector
	ETA 10	Vector + parallel
	CRAY YMP-8	} Vector + Shared Memory Parallel
	CRAY C90-16	
	CRAY T3E900	Scalar + MPI Parallel
2002	NEC SX6	Vector + MPI + Shared Memory Parallel

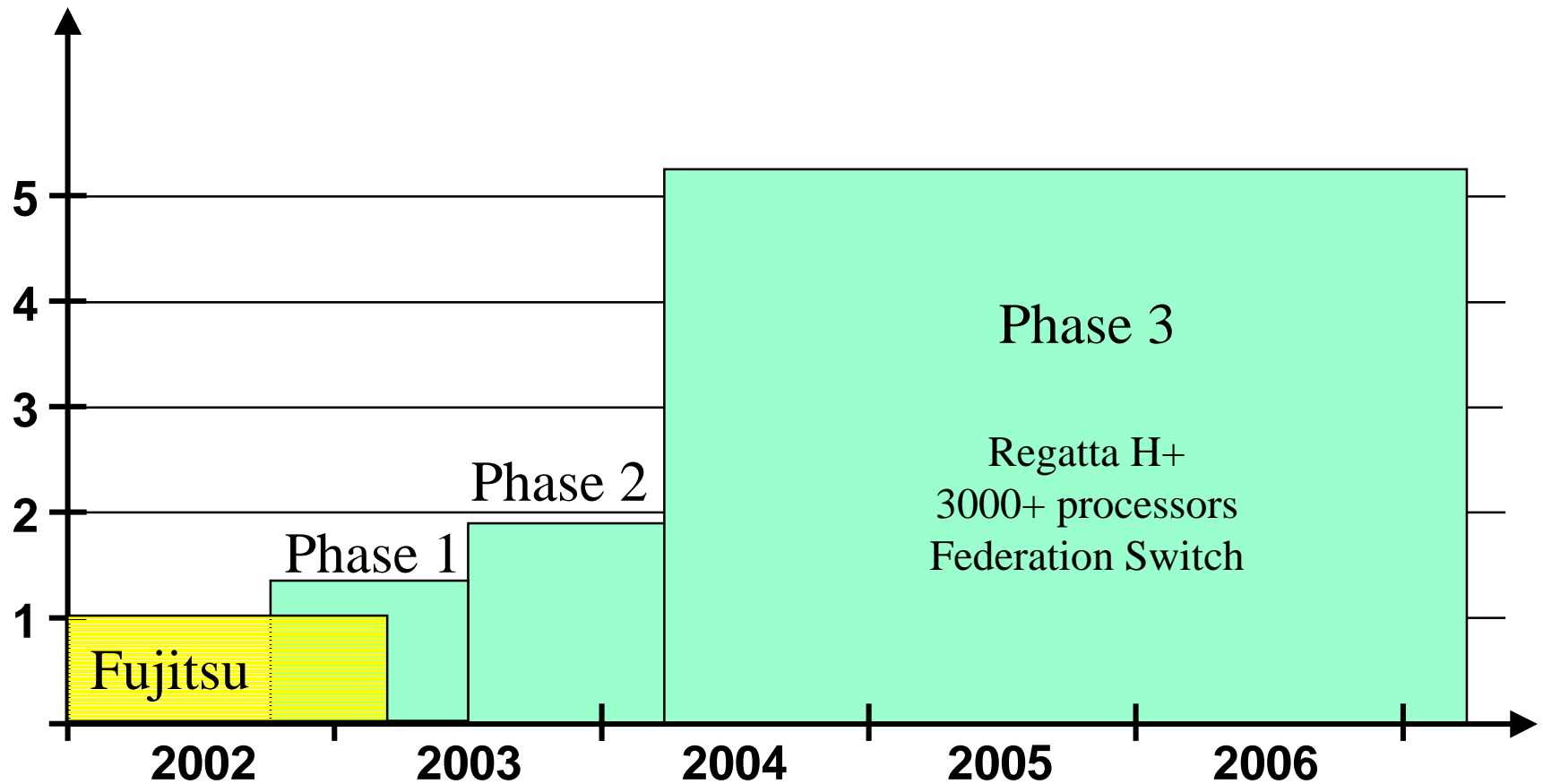
ECMWF supercomputer procurement in 2001

- Phased installation from 2003-2006 - parallel run early 2003
- Benchmark tests (for performance commitments)
 - 4D-Var (T511/255 - 2 copies)
 - T799 Forecast (2 copies)
 - T399 EPS (50 copies)
- Percentages of peak:
 - High Res. forecast & EPS scaled better than 4D-Var
 - ~10% for scalar architectures and ~30% for vector architectures

Outcome of the procurement

- Strong competition - with three highly-competitive tenders
- Offer from IBM judged to be best
- Contract runs until 31 March 2007 - all equipment on lease
- Key features :
 - Two identical clusters
 - Initially, p690 servers LPARed into 8-PE nodes
 - Later, p690 follow-on servers with the Federation switch

Performance profile of the contracted IBM solution relative to existing system



Performance on ECMWF codes (Fujitsu = 400 GF sustained)

ECMWF supercomputers

VPP to IBM

1999

2002

Fujitsu VPP5000
100 Vector processors

IBM p690
2 x 960 Scalar processors

Peak performance
9.6 Gflops per processor

Peak performance
5.2 Gflops per processor

8 processors per
shared memory node

4 GB memory per CPU

8 GB memory per node

ECMWF supercomputers

IBM

Phase 1 (2002)

IBM p690
2 x 960 processors

Phase 3 (2004)

IBM p690+
2 x 1600 processors

Peak performance
5.2 Gflops per processor

Peak performance
~7 Gflops per processor

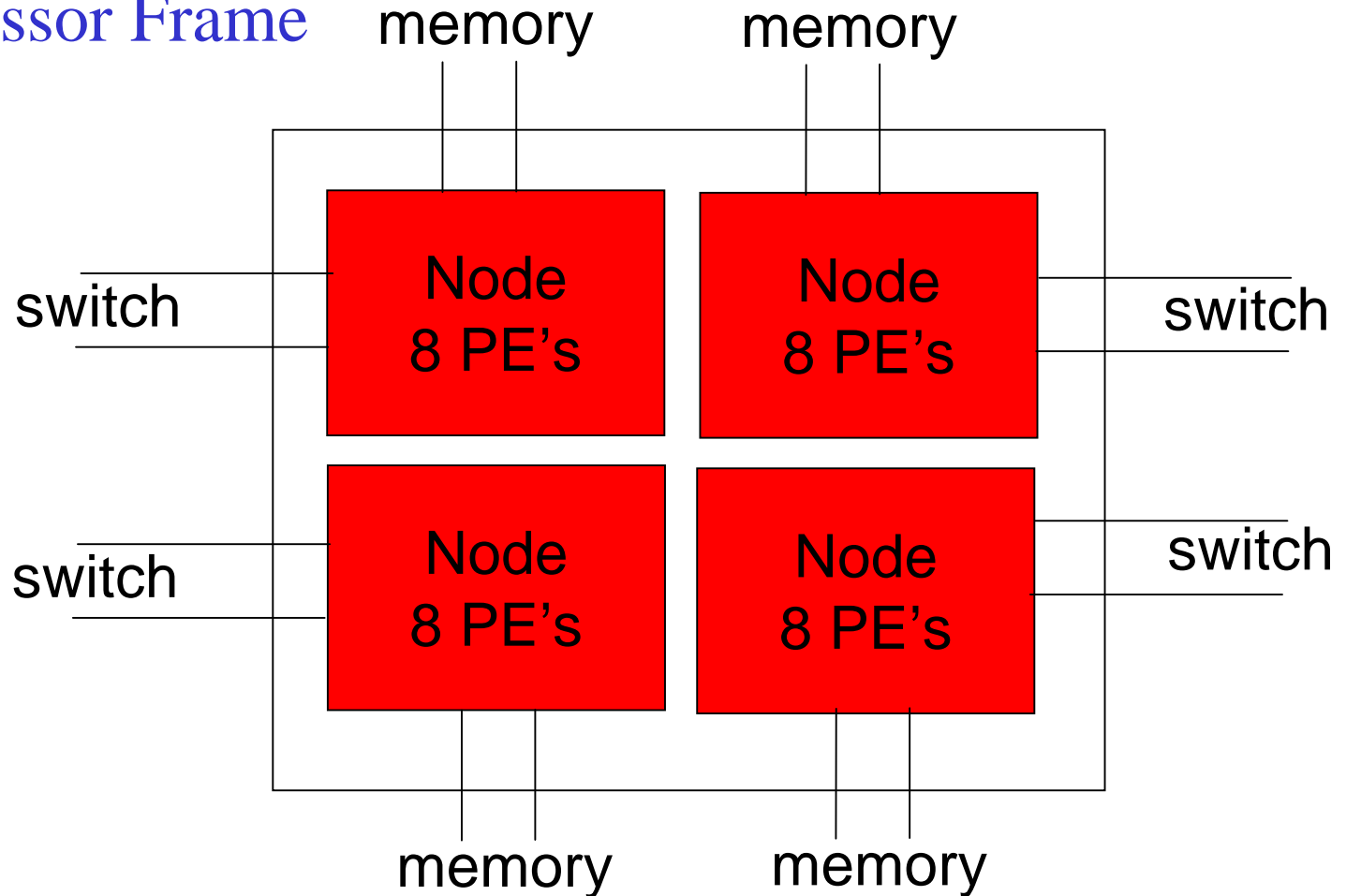
Switch 350 Mbytes/s

Switch >>350 Mbytes/s

8 processors per
shared memory node

IBM p690 LPARs

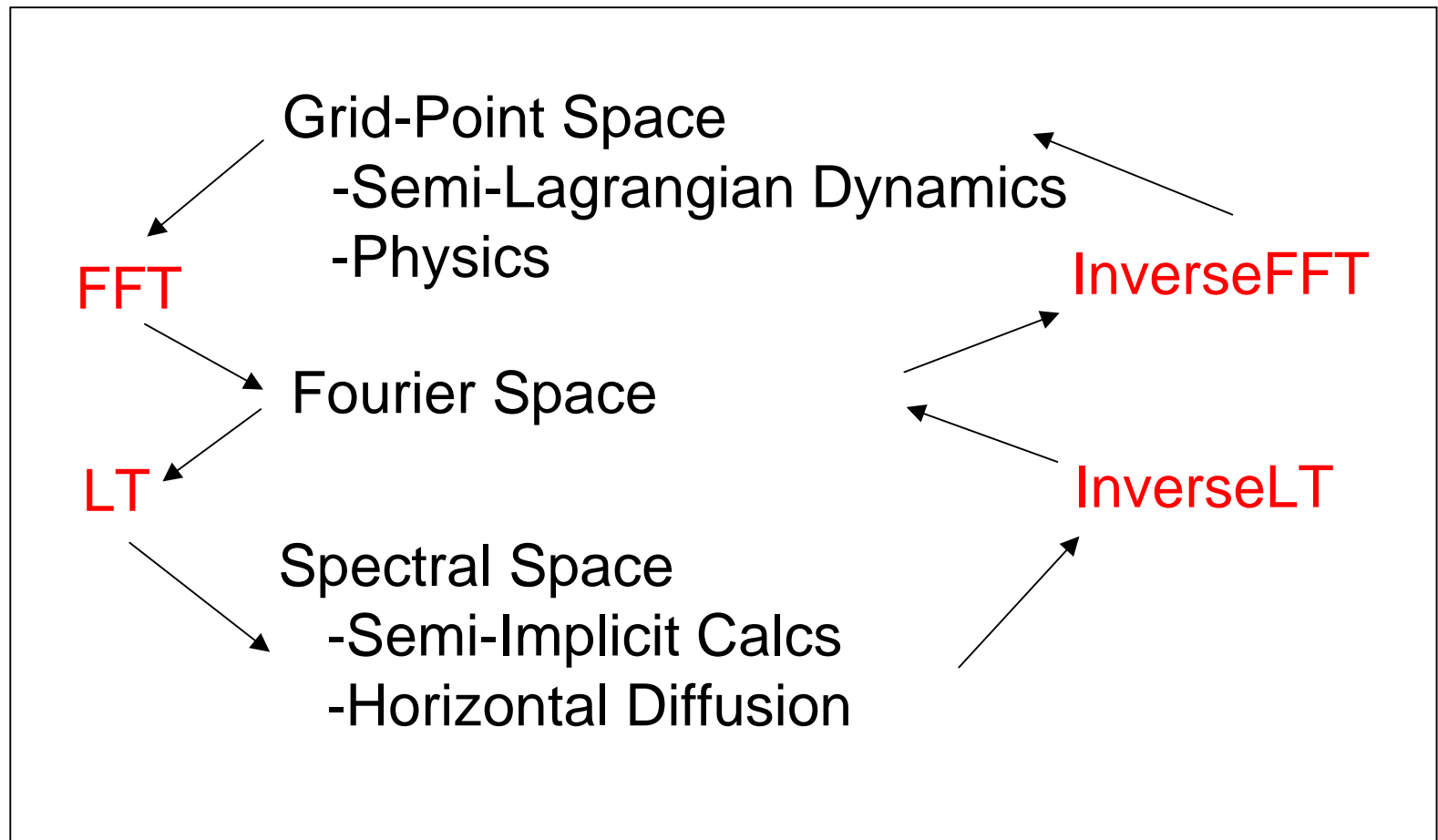
32 processor Frame



IFS system overview

- Spectral Model of the Atmosphere
- Developed jointly at ECMWF and Météo France
- More than 10 years old
- ~5000 routines, ~500,000 lines of source code
- Fortran 90 and some C
- Parallel using MPI and OpenMP
- Message passing characterized by large messages
- Rather flat execution profile,
- Matrix multiply is only standard maths library routine

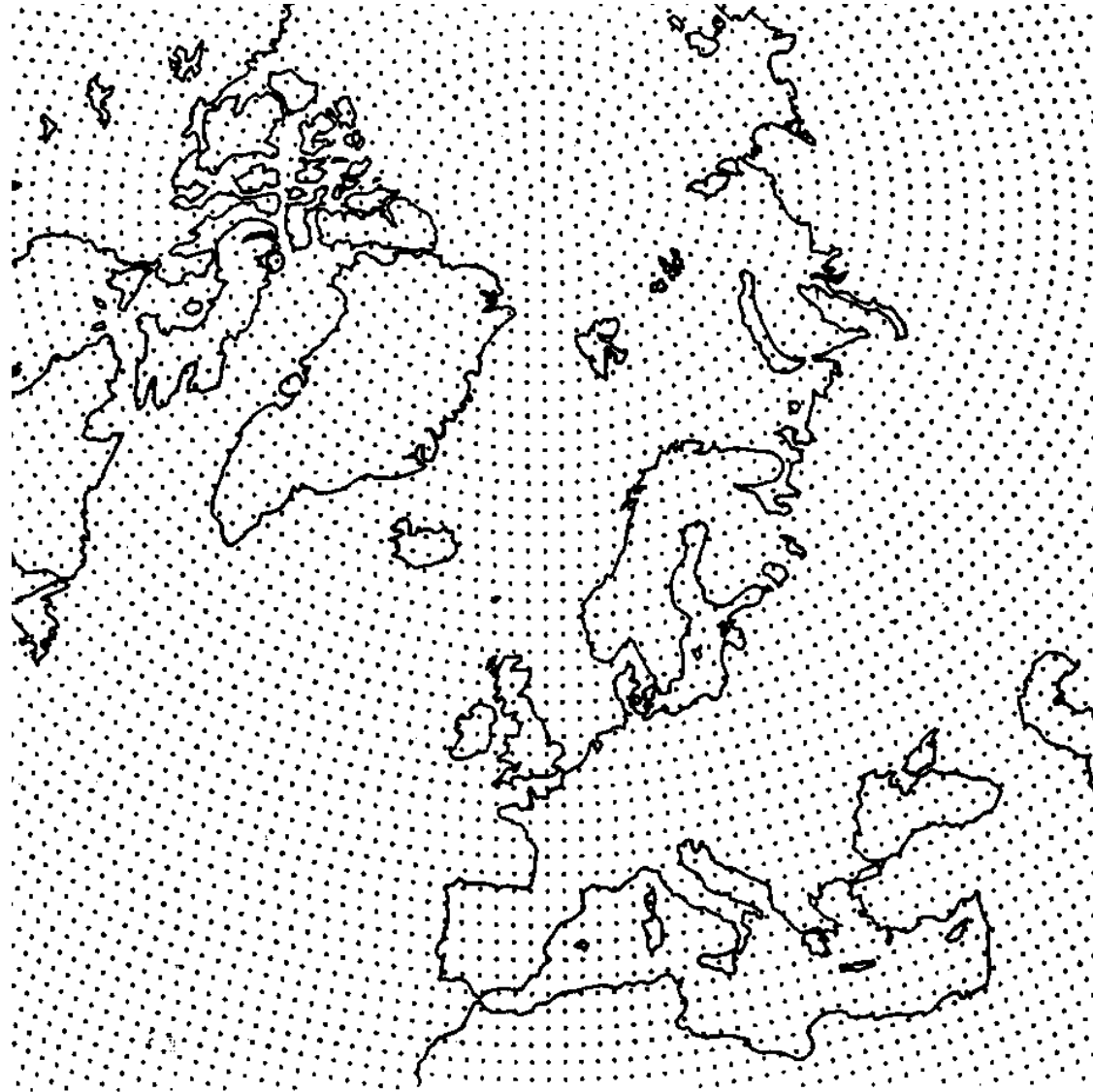
IFS – Spectral Model



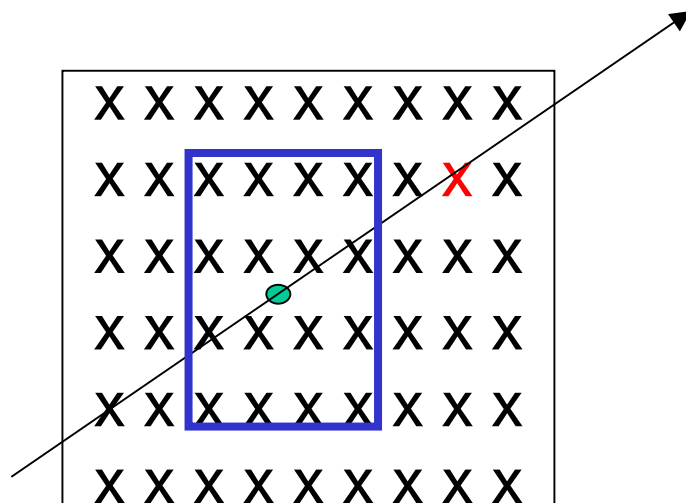
IFS Optimisation through changes to Algorithm

• Reduced Grid	x	1.4
• 2-time-level Semi-Lagrangian	x	10
• 'Linear Grid'	x	3.5
<hr/>		
TOTAL	~	50

Reduced Grid



IFS – Semi-Lagrangian Advection



Horizontal section

x arrival point at $t(n)$

• departure point at $t(n-1)$

— interpolation stencil

Full interpolation in 3-D is 32 point

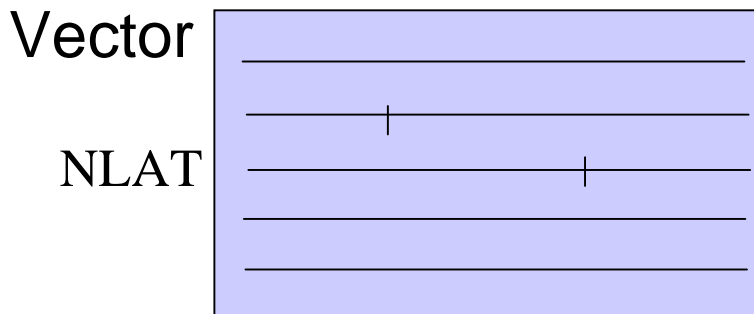
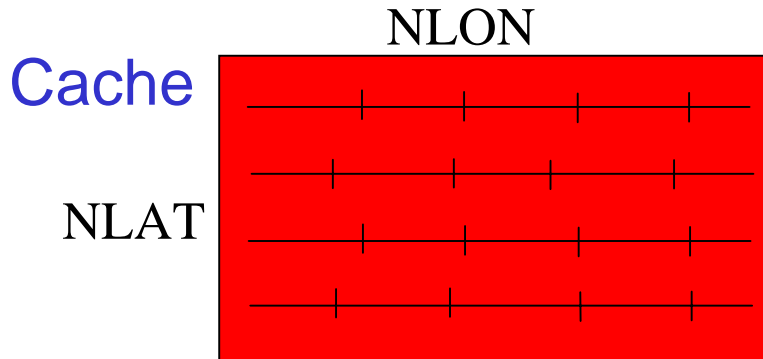
Requirements for ECMWF

- Error trapping always on
- Code must continue to run efficiently on vector machine
- Bit-wise reproducibility on different number of processors => -O3 -qstrict
- 64 bit arithmetic and 64 bit addressing

Design of IFS for vector, scalar, MPP and SMP systems

- NPROMA (cache and vector length)
- MPI (2-D decomposition)
- OpenMP
- Same code for all systems
–with few exceptions
- On-demand semi-Lagrangian communications

Grid-Point Calculations



```
DO J=1, NGPTOT, NPROMA
```

```
CALL GP_CALC
```

```
ENDDO
```

Lots of work
Independent for each J

High Level Blocking Scheme :

$U(NGPTOT, NLEV)$

$NGPTOT = NLAT * NLON$

$NLEV = \text{vertical levels}$

```
SUB GP_CALC
```

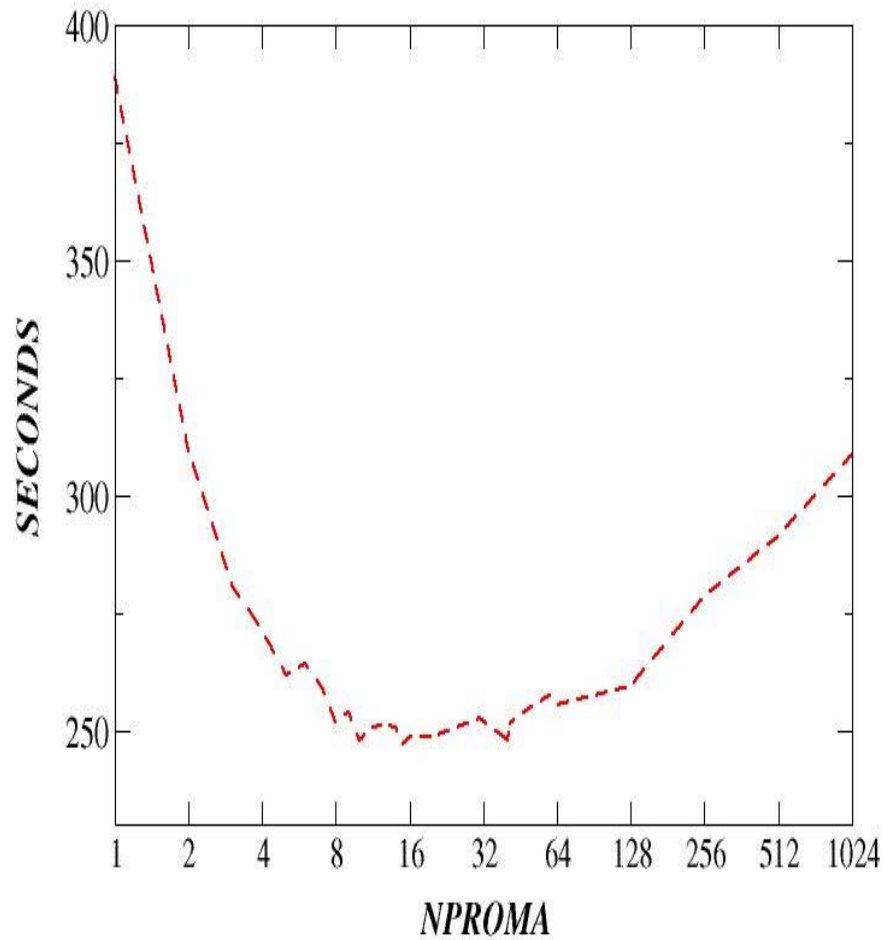
```
DO I=1, NPROMA
```

```
ENDDO
```

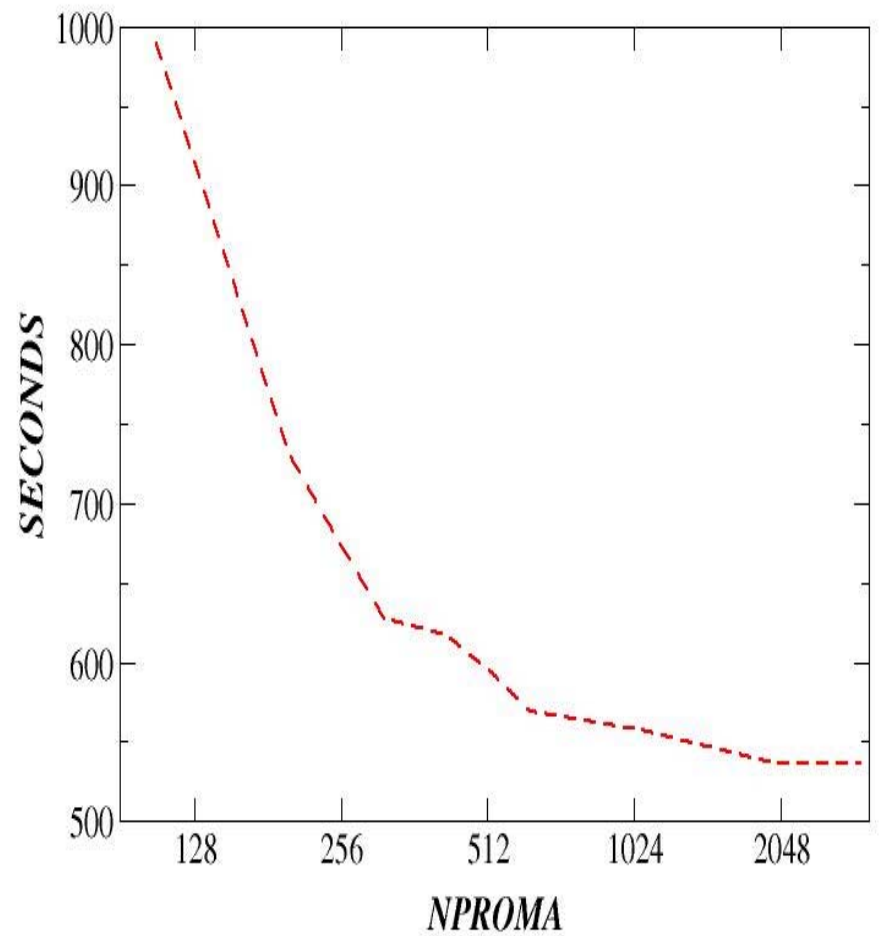
```
END
```

NPROMA

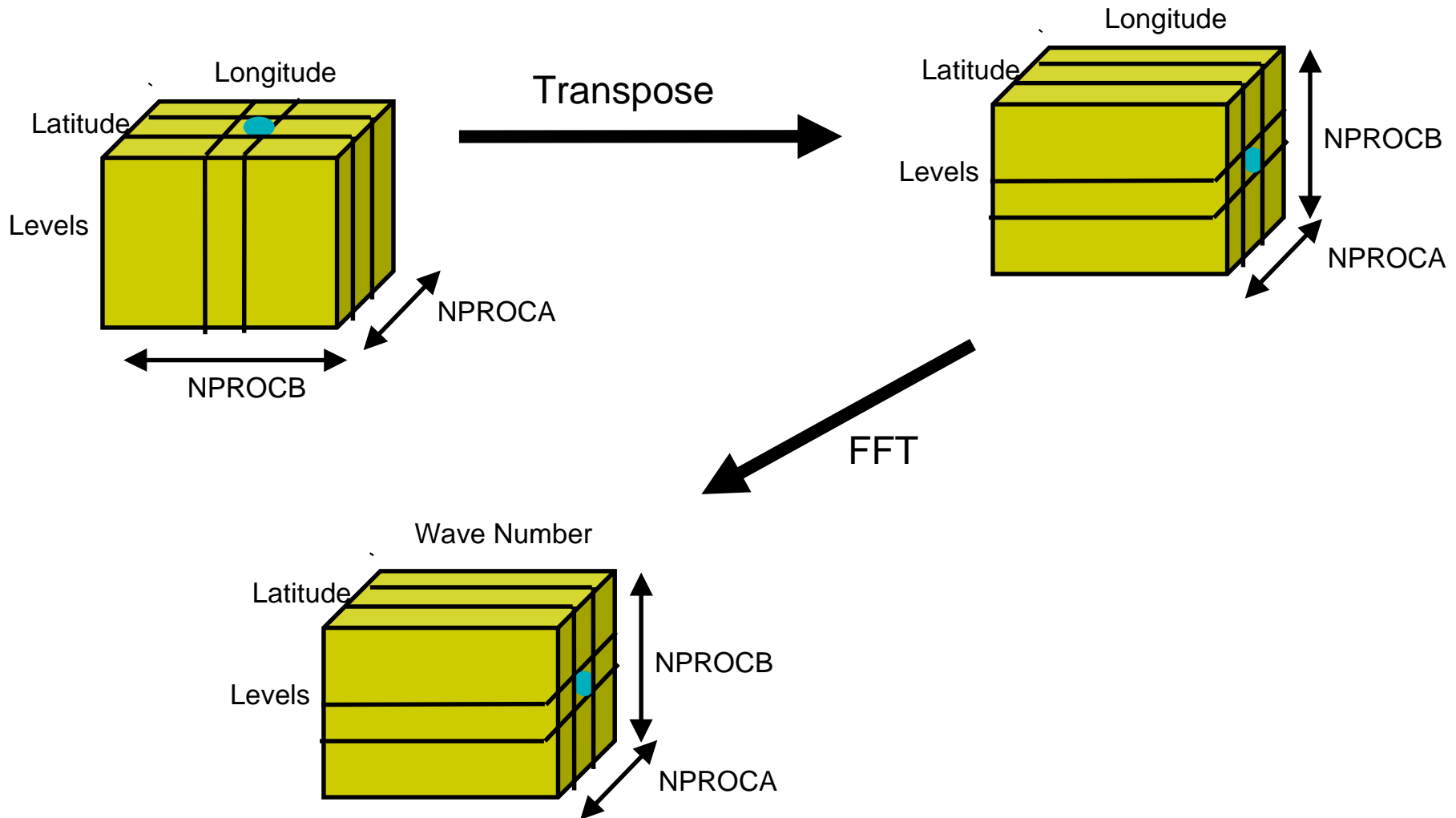
IFS T159 L60 : IBM p690 / 8



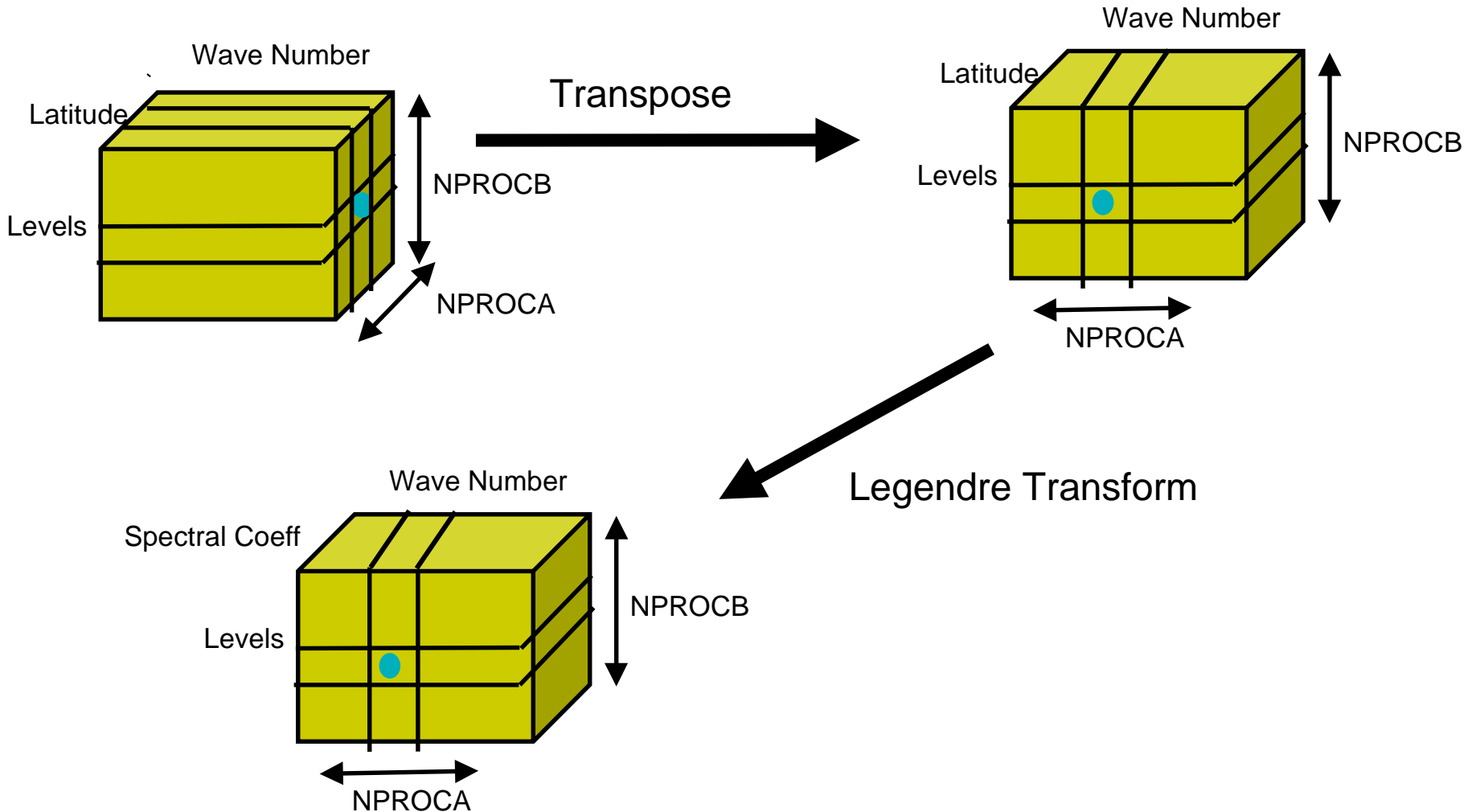
IFS T159L60 : vpp5000/1



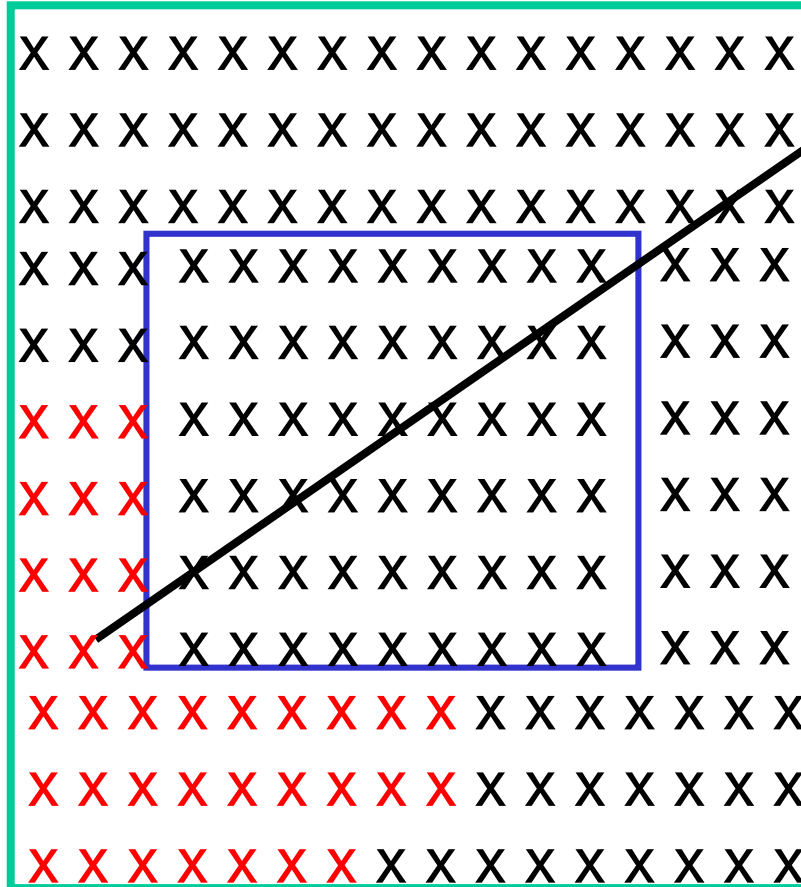
IFS: Transpositions



IFS: Transpositions



IFS – Semi-Lagrangian 'On Demand'



x points needed in halo

— core points on processor

— halo

OpenMP – parallelisation for shared memory

```
!$OMP DO PARALLEL PRIVATE(J)  
DO J=1, NGPTOT, NPROMA  
    CALL GP_CALCS  
ENDDO
```

- OpenMP parallel regions where old CRAY macrotasking was ('sort of')
- Need **thread safe** code inside a parallel region
- Not auto-parallel – by directive only

Why OpenMP in addition to MPI ?

PROS

- Improved performance
- Reduce MPI communications particularly SLCOMM and Global comms
- Reduce memory use

CONS

- Code maintainability
- Bugs can lurk unknown
- Must be thread safe

OpenMP techniques – ‘events’

Adjoint of semi-Lagrangian interpolation sums into same arrays from different threads

```
LOCKS set to 1

!$OMP DO PARALLEL
DO K=1,NGPTOT, NPROMA

    CALL BIG(K)

ENDDO
!$OMP END DO PARALLEL

SUBROUTINE BIG(K)

    CALL WAIT_EVENT(K,LOCK(1))

    update U

    CALL INCR_EVENT(NPROMA,LOCK(1))
    CALL WAIT_EVENT(K,LOCK(2))

    update V

    Etc etc.
```

OpenMP techniques – ‘events’

Get ‘stair-case’ effect :

SUBROUTINE BIG(K)

CALL WAIT_EVENT(K, LOCK(1))

update U



1 2 3 4

CALL INCR_EVENT(NPROMA, LOCK(1))

CALL WAIT_EVENT(K, LOCK(2))

update V

thread numbers

1 2 3 4

Etc etc.

1 2 3 4

1 2 3 4

1 2 3

OpenMP techniques – ‘events’

Finite difference code with ‘DO PARALLEL’ at high level

```
! LOCK ARRAY set to 0
```

```
!$OMP DO PARALLEL  
DO IT=1,NPES
```

```
DO K=1,KE  
  DO J=J1,J2  
    DO I=1,IE  
      B(I,J,K)=  
    ENDDO  
  ENDDO  
ENDDO
```

```
! LOCK ARRAY SET TO 1  
CALL INCR_LOCK(IT,LOCK)  
CALL CHECK_LOCK(1,IT,LOCK)
```

```
DO K=1,KE  
  DO J=J1,J2  
    DO I=1,IE  
      A(I,J)=A(I,J) &  
      & +B(I,J+1,K)+B(I,J-1,K)  
    ENDDO  
  ENDDO  
ENDDO
```

Etc etc.

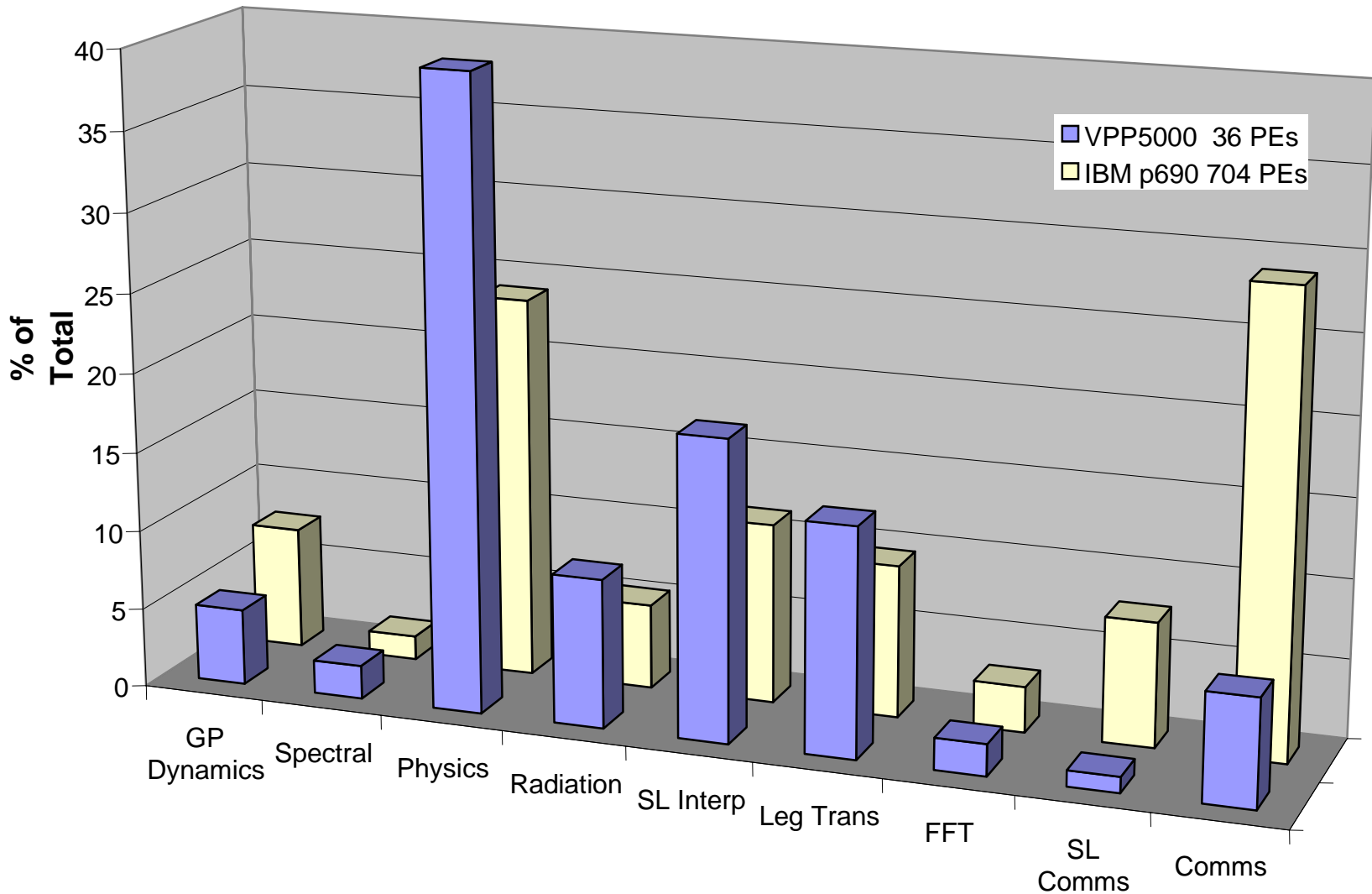
```
ENDDO  
!$OMP END DO PARALLEL
```

Performance

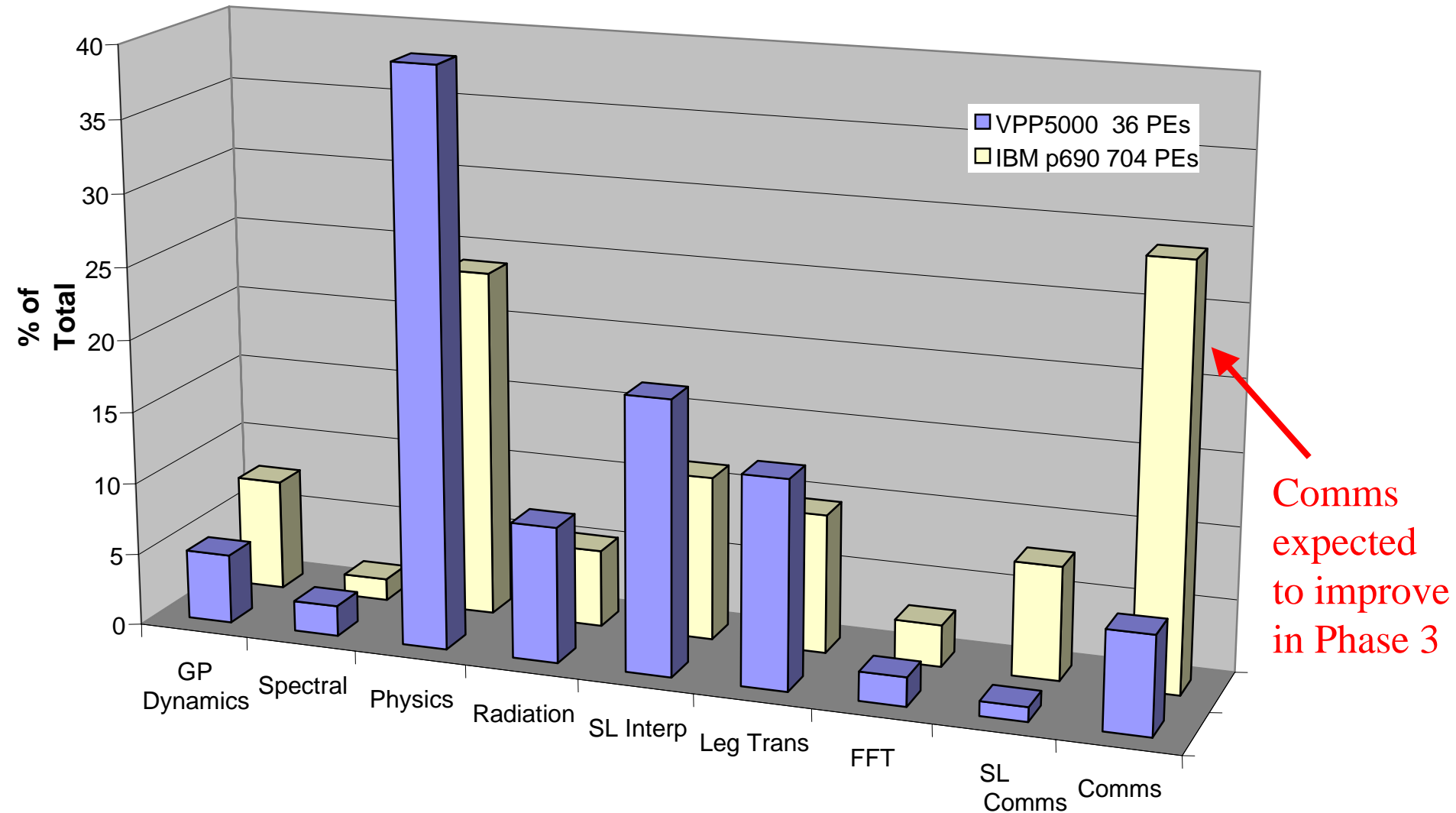
Profiles, scalability and comparison of
IBM p690 with Fujitsu VPP5000

- Phase 1 benchmark
- Latest IFS cycle

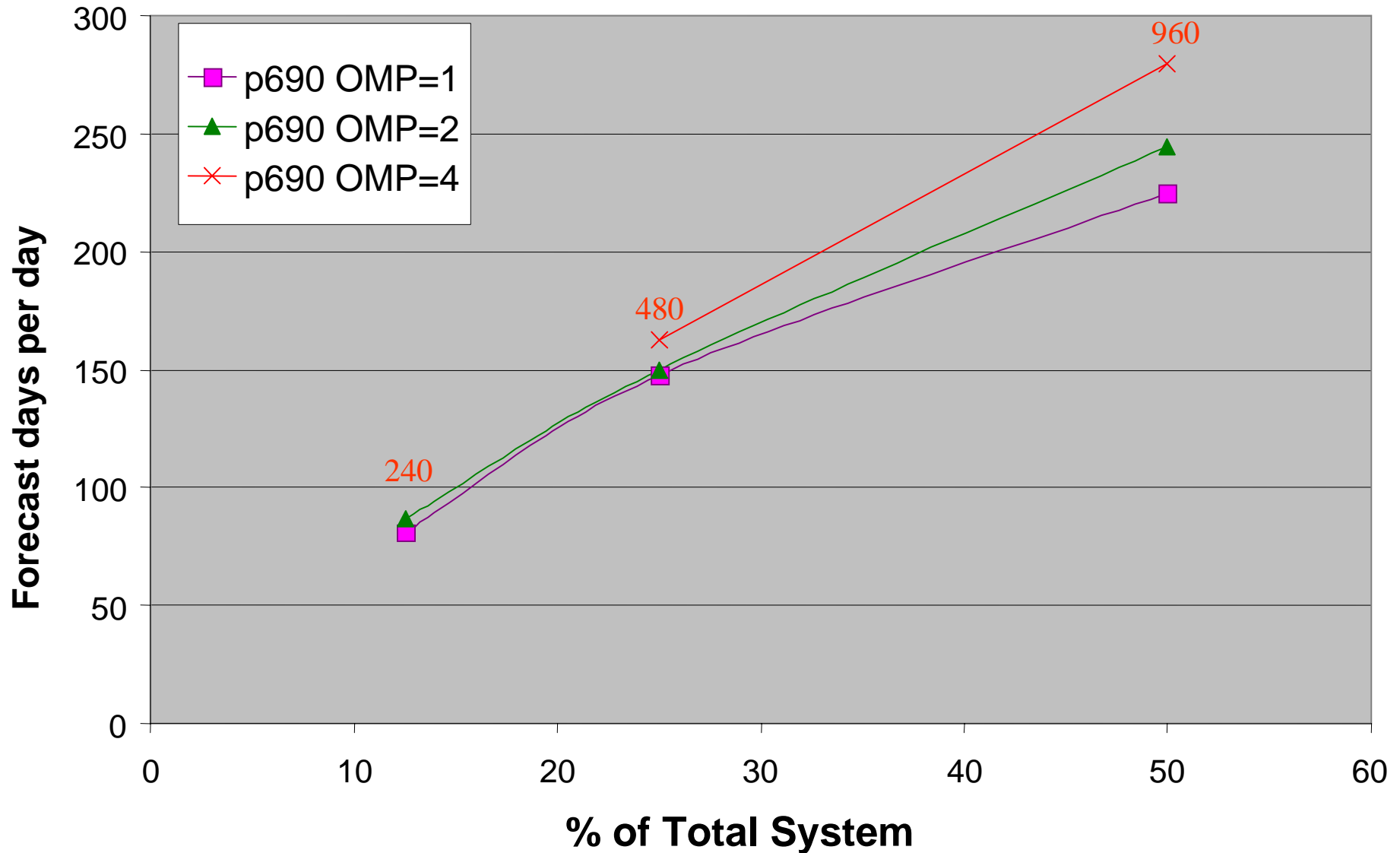
High Resolution Forecast - T799 L90



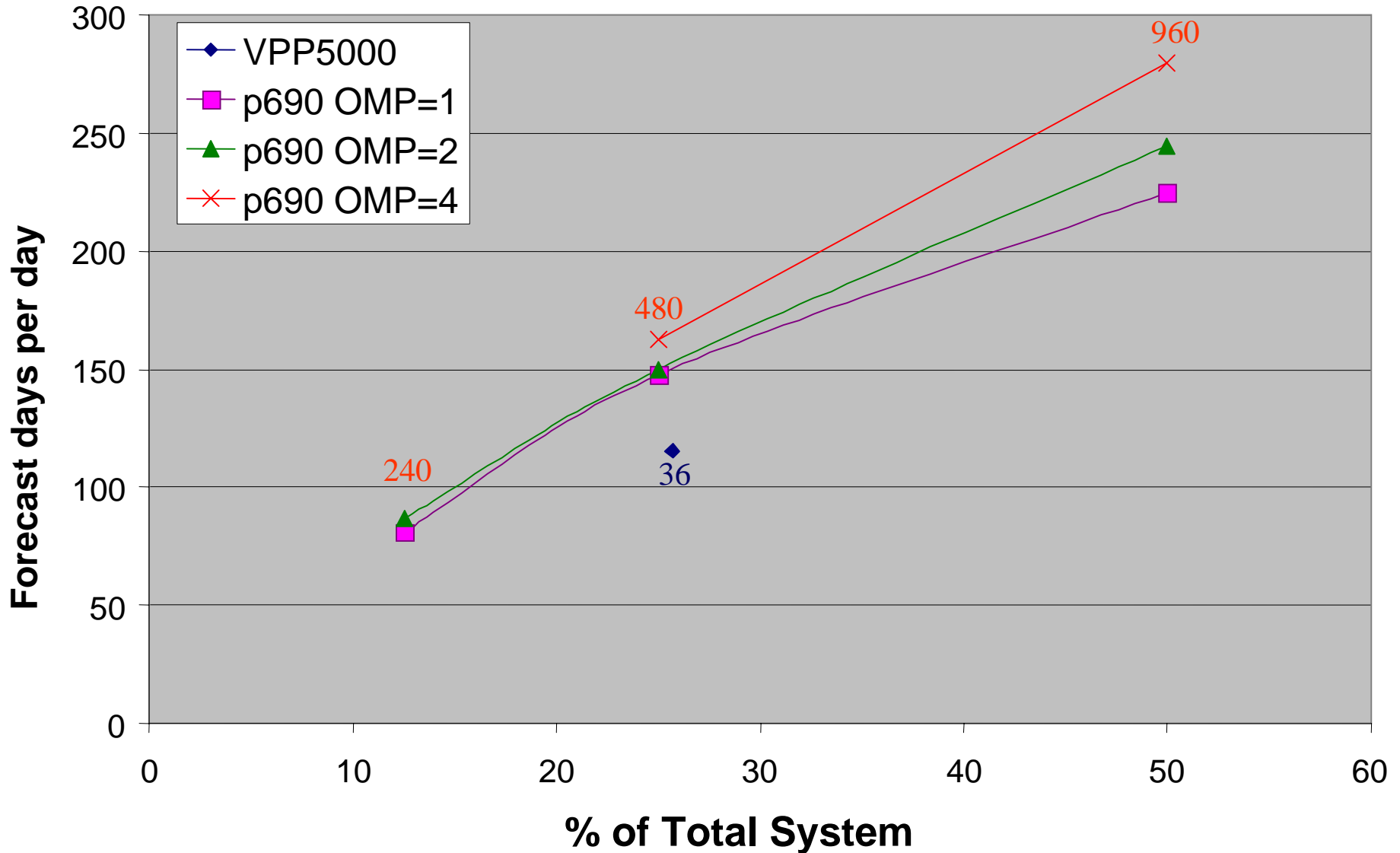
High Resolution Forecast - T799 L90



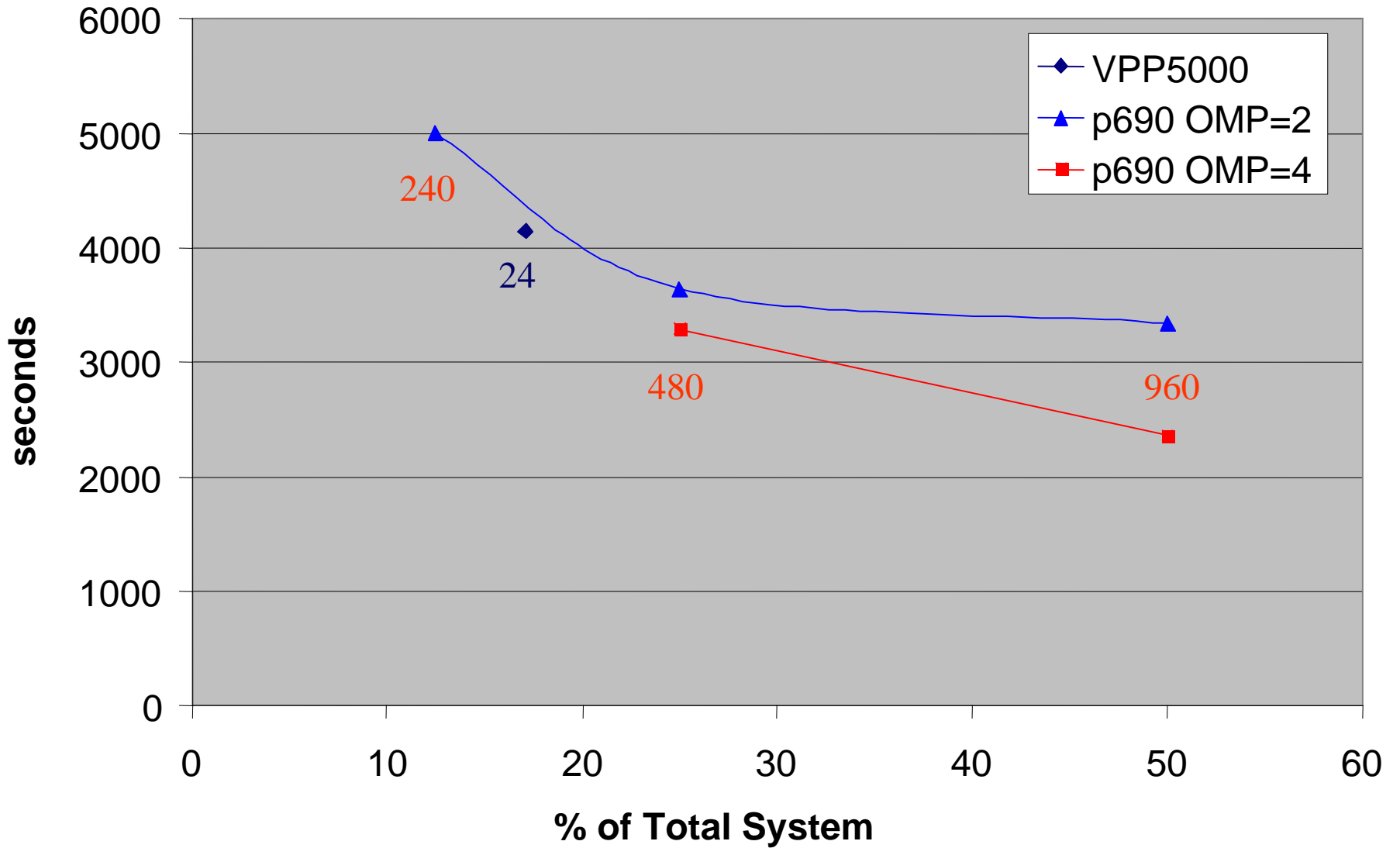
High Resolution forecast : T799 L90



High Resolution forecast : T799 L90



4D-Var : T511/T255 L60



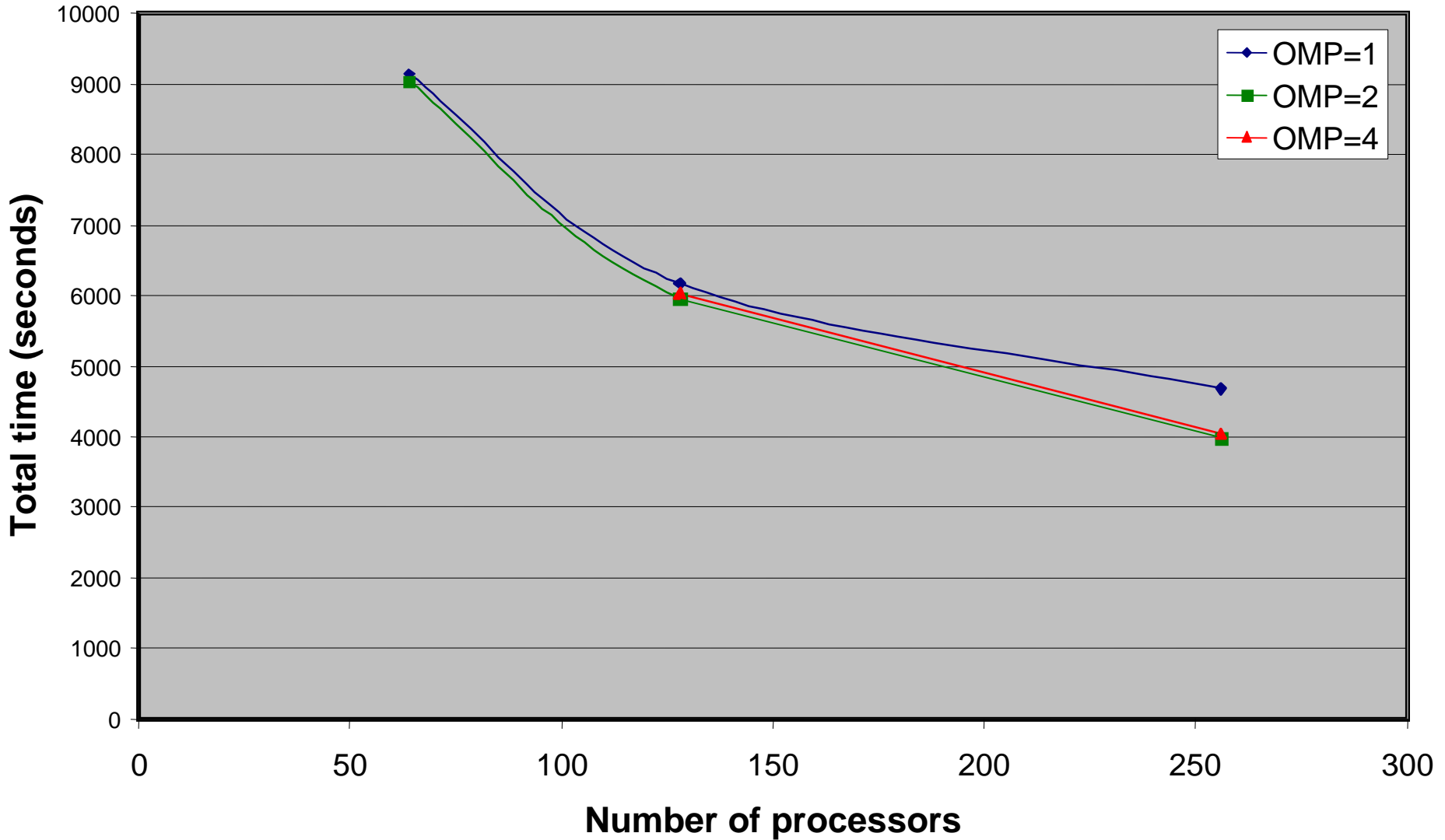
4DVar T511/T159 L60 using latest IFS cycle (25r4)

	VPP5000	IBM p690
Processors	16	128 (64 MPI & 2 OMP)
% of total system	11%	6%
Run time	1 : 24	1 : 39
Memory	3.5 GB per CPU	3.8 GB per node
How many p690 PE's = 1 VPP5000 CPU?		9.4

4DVar T511/T159 L60 using latest IFS cycle (25r4)

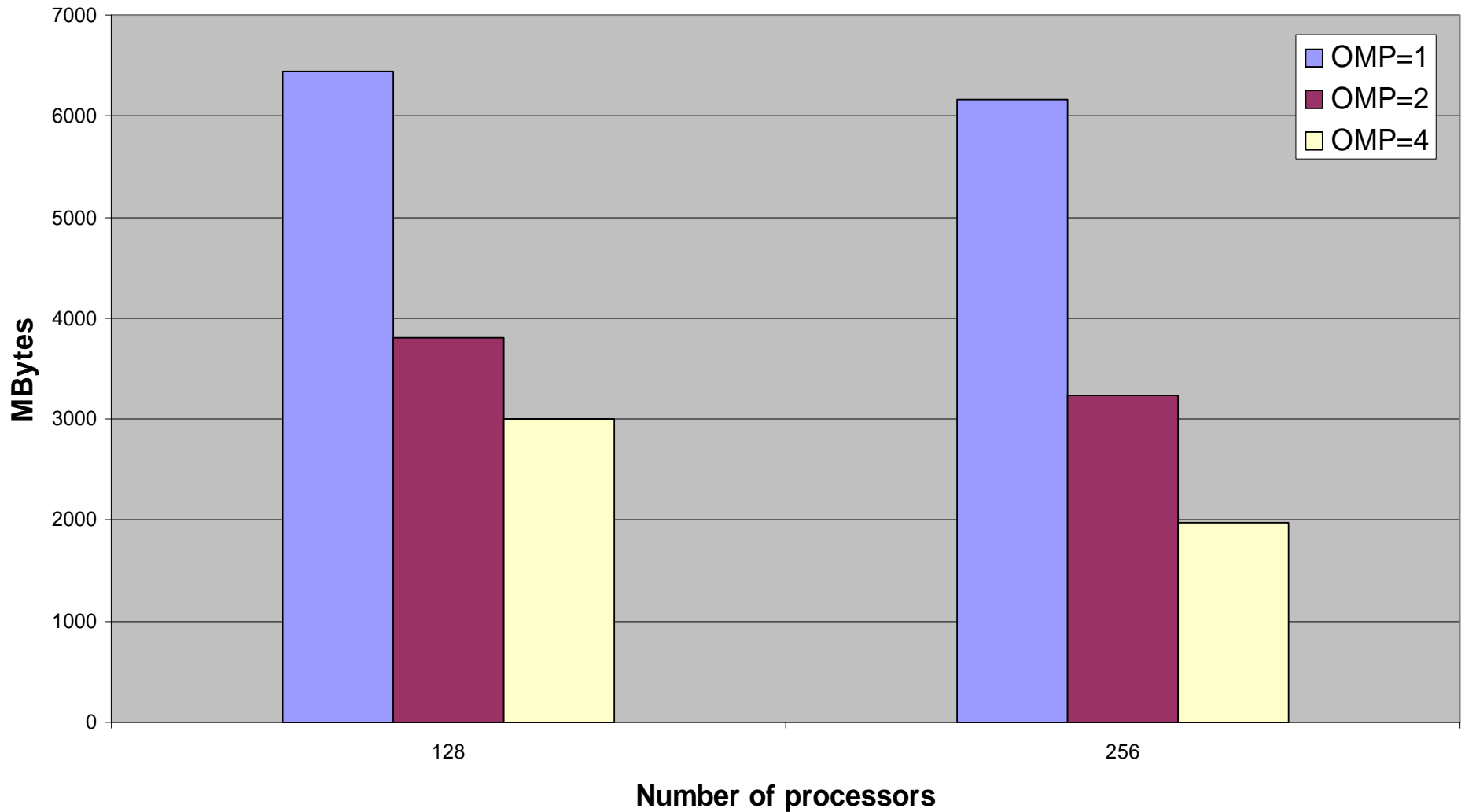
	VPP5000	IBM p690	
Processors	16	256 (256MPI & 1 OMP)	256 (128 MPI & 2 OMP)
% of total system	11%	13%	13%
Run time	1 : 24	1 : 18	1 : 06
Memory	3.5 GB per CPU	6.2 GB per node	3.2 GB per node

4D-Var T511/T159 L60 on IBM p690



4D-Var T511/T159 on IBM p690

Memory per Node



Optimisation

- Detailed timing
- Message passing
 - use global MPI routines
 - don't overlap buffer packing/unpacking with comms
- OpenMP – more parallel regions
- Compiler : xlf90_r with -O3 -qstrict
 - Remove copies, temporary arrays and zeroing of arrays
 - Recode divides & array syntax
 - Common expressions in brackets